```
icon class cluster(r)
begin
maximum size is (110,60);
position is
   (case r.type begin
                "CV"     :800
                "SSN"    :1600
                "SSBN"   :1600
                "SSGN"   :1600
                "CGN"    :2500
                "CG"     :2500
                "CA"     :2500
                "DDG"    :3200
                "FF"     :3200
                "AGI"    :3200
                "AO"     :4000
                default  :1600
                end,      case r.nat begin
                          "US":1200
                          "UR":2000
                          default:2500
                          end );
      template icon  case r.type  begin
                                  "CV"     :carrier
                                  "SSN"    :sub
                                  "SSBN"   :sub
                                  "SSGN"   :sub
                                  "CGN"    :cruiser
                                  "CG"     :cruiser
                                  "CA"     :cruiser
                                  "DDG"    :destroyer
                                  "FF"     :destroyer
                                  "AGI"    :destroyer
                                  "AO"     :oiler
                                  default  :cruiser
                                  end;
      scale is r.beam*2 percent;
      color of region 1 is case r.ready begin
                                        "1":green
                                        "2":yellow
                                        "3":orange
                                        "4":red
                                        default: grey
                                        end ;
      attribute region r.nam from ( 5,16) to (70,28);
      attribute region r.Ircs from ( 5,28) to (70,40);
      attribute region r.conam from ( 5,40) to (70,52);
end;
```

Figure 15. Icon Class Description

A data surface is populated with icons in response to the user typing a display statement to the query language of the database management system. This statement specifies which entities are to be represented as icons. Through an optional qualification to the associate statement, the user may specify that only certain entities be selected, such as only U.S. ships with less than 20% fuel. For each entity which is selected, the system creates one icon by interpreting the icon class description with its variables bound to values of the particular entity being displayed.

The icons themselves are not stored in the symbolic database but are graphical interpretations of data contained in that database. This is in contrast to systems like [12] where a symbolic database management system is used to contain descriptions of the geometric primitives used to construct a picture. The approach taken by SDMS allows the system to display shared databases which do not originate as pictures in SDMS. The statements in an icon class description perform some graphical action and take arguments whose values are attributes of the entity being displayed.

The position statement determines the placement of the icon on the data surface. In the example, it maps the ship's type into x-coordinates and nationality into y-coordinates. Once an icon is created, further attempts to create icons at that location result in a nearby location being used.

The template statement specifies the shape of the icon by selecting among a set of pictures which have previously been drawn by the database administrator.

The scale statement specifies the size of the icon. In the example this is a function of the beam of the ship.

The color statement specifies the color of each ship according to its readiness.

Finally, the three attribute region statements place the values of the ship's name, its international radio call sign, and the commanding officer's name into the specified locations in the icon.

The use of templates allows the user to define complex icon shapes without requiring the use of an awkward graphics language. Different templates and icon class descriptions may be defined for each level of detail in the data surface, allowing explicit control over the scaling process. For instance, the ship data surface presents different amounts of text and different pictures of ships at each level of detail. The template for one type of ship is shown below. Note the four different templates, one for each level of detail in the ship data surface.

Figure 16. Templates

A template is the starting point for the generation of an icon. Its shape and color determine the shape and color of the icon. A template is divided into regions which are distinguished by the colors with which they are drawn. This allows arbitrary complexity in the shape of regions. The color of a region may be transformed by a color statement in the icon class description. For example, the colors of the ships themselves in the ship data surface are displayed as a function of the readiness attribute of that ship in the database.

4. IMPLEMENTATION

The SDMS prototype runs on a dedicated PDP-11/70 running a modified version of the UNIX operating system [13]. The machine has 1.25 megabytes of memory which is used primarily for manipulating the graphical data surfaces and staging them to the display. The operating system has been modified to allow user programs to map selected portions of their 16 bit address space into portions of physical memory which contain graphical data. The images are displayed on a Lexidata raster scan display system which refreshes three independent color displays out of its own 480x640 frame buffers. The SDMS system runs in several parallel processes which communicate through shared memory and pipes (a low-speed interprocess channel which appears to a user program as an i/o device). The prototype configuration supports a single user. Although general purpose time sharing can be run simultaneously with SDMS, memory address space limitations preclude supporting more than one SDMS user simultaneously.

Data surfaces are stored on a moving-head disk as bit arrays known as image-planes. An image plane is partitioned into tiles which are rectangles of 64 lines, each containing 128 pixels. Thus each tile occupies 8K bytes, which is equal to the page size of the PDP-11/70. The ships database, shown earlier, consists of four image planes (to support zooming to four levels of detail) and occupies 36 megabytes.

A simplified process structure appears below. All of the processes shown run in the 11/70. The various input devices connect directly to the 11/70 where they are read by processes indicated on the diagram.

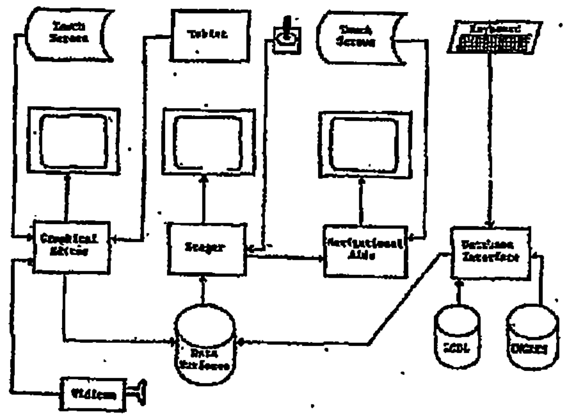


Figure 17. SDMS Process Structure

The stager is actually two processes which allow the user to move the display window over the data surface. One of these processes reads the bit-array representation of the data surface into the main memory of the PDP-11/70. This process endeavors to keep in memory all tiles currently on the screen plus sufficient extra tiles to maintain the user's current direction of motion. A second process sends portions of these tiles to the display, once again endeavoring to maintain a suitable margin of data just off-screen to allow for motion across the data surface. When the user presses on the joy stick, indicating a desire to move in a given direction, this process first checks to make sure that sufficient picture data is loaded into the display to start moving in that direction. If so, the display is commanded to begin its screen refresh at a new location, causing the picture to scroll. The program then determines whether new data must be sent to the display to maintain the margin around the displayed picture. This may require that additional data be read from the disk as well. This process continues as long as the user holds down the joy stick or until an edge of the data surface is reached. This same program is responsible for updating the global variables describing the user's current position.

These two programs are also involved in zooming the display. When the user twists the joy stick, the display is commanded to change its scale factor. If the user is over a port or in a data surface which has multiple image planes, the stager must prepare to display the new, more detailed image plane. Fortunately, zooming the display reduces the storage required to contain the current image. The memory which remains is filled with the new image plane in anticipation of the user's continued twisting of the joy stick. Finally, if the user does continue to twist the joy stick, the next image plane, which has already been loaded into the display memory, appears on the screen.

The navigational aids process is responsible for displaying the current world-view map or the data surface hierarchy map. It alters the position of the highlighted rectangle as the global values of the user's current coordinates change. This process also listens to the touch sensitive digitizer mounted in front of the navigational aid display and directs the stager to perform a rapid transit operation if the user touches the screen.

The database interface is responsible for creating data surfaces in response to user commands and for blinking selected icons when so requested. It is also responsible for maintaining the consistency between the symbolic and graphical representations in the database. To this end, it monitors updates to the symbolic database. If such an update changes the value of an attribute which was used to create an icon, that icon is recreated with the new data.

The graphical editor allows the user to modify the graphical data space. It operates upon the same in-core tiles used by the stager. It also manipulates the system's description of the graphical data space, allowing the user to delete icons, designate pictures as templates, and create ports.

5. CONCLUSIONS

The initial informal evaluations by users who have used the system in our laboratory have been enthusiastic. Most people learn to use the controls in a matter of seconds and can create pictures almost immediately. Much of this success can be attributed to the small number of simple controls and the furnishing of immediate feedback to every user action. The touch sensitive digitizers simplify the process of directing user input to the appropriate display. They would have been used for all three displays (replacing the use of the data tablet in the graphical editor) if they had sufficient resolution and a means of sensing finger position (for intermediate feedback) before the user pressed on the screen.

The SDMS method of accessing data seems to be most appropriate for those situations where the user needs to browse through a database. Since the data to be retrieved need not be specified precisely, the user does not need to learn a formal query language or possess an intimate knowledge of the structure and contents of the database. On the other hand, SDMS is not especially well suited to those circumstances which require locating some number of entities in the database which meet some precisely stated criterion.

As was anticipated, the least successful aspects of the system are those which require keyboard input. The selection of entities which are to appear on a data surface requires the use of a database query language which is awkward at best. Likewise, the icon class descriptions require the services of someone who not only knows the organization of the database and the various commands but has the talent to design a data surface which is useful and aesthetically pleasing.

A more rigorous evaluation of the system will take place during the coming year when the system is installed at several selected sites where it can be used under conditions more closely approximating those of the ultimate operational environment. This process will involve loading the system with real databases and adding process ports for some previously implemented decision analysis programs.

5.1 Future Work

The usefulness of the system could be greatly increased if graphic representations of databases could be created with less human intervention. An approach outlined in [11] would use profiles of users and their applications together with descriptions of the semantics conveyed by various representations to select the most appropriate representation for any particular circumstance.

The composition of complex queries could be accomplished by allowing the user to manipulate the graphical representations, creating a picture of the desired result in a form of graphical query-by-example. This might make the requisite logical thinking less painful for the user.

In terms of implementation details, two very specific goals will be pursued: creating icons more quickly and storing graphical data surfaces more compactly. A much faster icon creation would allow the icons to be generated on the fly as the user scrolled over the data surface. This would greatly reduce the storage requirements of the system and also eliminate the problems of maintaining the consistency of the two different methods of storage currently employed. One approach which will be attempted in the short term is to split the process of icon creation into two stages. The retrieval from the symbolic database will produce a "compiled" icon description. This description will be expanded as the user traverses the data surface. This approach would allow several users to examine the same graphical data space in an efficient and economical manner.

References

1. Hendrix, G., Sacerdoti, E., Sagalowicz, D., Slocum, J., "Developing a Natural Language Interface to Complex Data", ACM Transactions on Database Systems 3:2, June, 1978.

2. Woods, W., Kaplan, R., Nash-Webber, B., The Lunar Sciences Natural Language Information System, Bolt Beranek and Newman, Cambridge, Mass., June, 1972.

3. Zloof, M., "Query by Example", Proc. AFIPS 1975 NCC, Vol 44, AFIPS Press, Montvale, N.J.

4. McDonald, N., Stonebraker, M., "CUPID: The Friendly Query Language", Proc. ACM Pacific Conference, San Francisco, April, 1975.

5. Donelson, W., "Spatial Management of Information", Proc. ACM SIGGRAPH 1978, Atlanta, Georgia.

6. Bolt, R., Spatial Data Management, Architecture Machine Group, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1979.

7. Robertson, G., McCracken, D., Newell, A. The ZOG Approach to Man-Machine Communication, technical report CMU-CS-79-148, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, October, 1979.

8. Stonebraker, M., Held, G., Wong, E., "INGRES - A Relational Data Base System", Proc. AFIPS, Volume 44.

9. Shoup, R., "Superpaint...the Digital Animator", Datamation, May, 1979.

10. Smith, A. Paint, Tech. Memo No. 7, Computer Graphics Lab, New York Institute of Technology, Old Westbury, NY, July, 1978.

11. Herot, C., Carling, R., Friedell, M., Kramlich, D., and Thompson, J., "Spatial Data Management System Semi-Annual Technical Report," Technical Report CCA-79-25, June, 1979.

12. Weller, D., Williams, R., "Graphic and Database Support for Problem Solving", Proc. ACM SIGGRAPH 1976, Philadelphia.

13. Ritchie, D., Thompson, K., "The UNIX Time Sharing System", Communications of the ACM, 17:7, July 1974.

THE MANAGEMENT OF VERY LARGE TWO-DIMENSIONAL RASTER GRAPHICS ENVIRONMENTS

Mark Friedell, Richard Carling, David Kramlich and Christopher F. Herot

Computer Science Research Division, Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

ABSTRACT

An operational information management system capable of managing very large two-dimensional raster graphics environments is described. A database computer, specially designed for image data, is proposed as an extension to the current system.

Algorithms for encoding, retrieving, and staging image data are discussed.

1. Introduction

The Computer Corporation of America's Spatial Data Management System (SDMS) [1] [2] is an experimental information management system capable of supporting a two-dimensional graphics environment orders of magnitude larger than those supported by other currently operational graphics systems. At CCA and elsewhere,[3] large environment graphics research and development efforts are recognizing the need for an image data management subsystem capable of operating in parallel with the image producing hardware.

The intent of this paper is to suggest the use of a database computer to manage the kind of two-dimensional environments supported by SDMS. There are two principle topics in the paper: Sections two and three discuss the image data management and manipulation algorithms in the currently operational SDMS, and section four proposes a design for a database computer image management subsystem.

2. SDMS Overview

SDMS allows a user to view pictorial representations of symbolic data, where the appearance of the representation is determined by symbolic attributes of the data. These pictorial representations are located in a graphical data space (GDS) which is accessed through a set of three color, raster-scan displays. One of the displays presents a "world-view" map of the entire data surface. A magnified view of the data surface is simultaneously presented on another display. The location of the magnified portion is indicated by a highlighted region on the world-view map. The user controls which portion of the data surface is displayed by means of a joystick. Pressing the joystick in any given direction causes the user's magnified window to move in that direction over the data surface.

The GDS consists of one or more separate data surfaces, called I-spaces, each of arbitrary size. Each I-space may, in turn consist of several data surfaces, called iplanes. Each iplane in an I-space presents a view of the image at a different resolution. The user selects which iplane is displayed by means of a knob atop the joystick which controls motion in the Z direction. Twisting the knob clockwise causes the next (more detailed) iplane below the current one to be displayed. Twisting the knob counterclockwise causes the previous (less detailed) iplane above the current one to be displayed.

The current implementation of SDMS runs on a PDP-11/70 under the Unix operating system. The hardware configuration includes moving head disks for storing image data and 1.25 megabytes of main memory. There are three 640 by 480 frame buffers driving the displays.

3. Staging Image Data

A central component of SDMS is the ability to scroll an image considerably larger than the display screen. The effect of motion is achieved by changing the portion of the image which is displayed on the main screen. Since the frame buffer is not sufficiently large to contain an entire image plane, this must be accomplished by moving data from the disk to the frame buffer. This process is referred to as staging.

The data stored in the frame buffer consists of the data appearing on the screen plus a small amount of surrounding data. A somewhat larger area is stored in the memory of the PDP-11/70. In the current configuration, the image is stored in its entirety on moving head disk drives. The nesting of these representations is illustrated in figure 3.1. Motion across an iplane requires staging data from the disk to the PDP-11's memory and then to the frame buffer in order to maintain the required nesting.

In order to manage an image this size in core, the image is partitioned into smaller-sized units known as tiles. The size of the tiles has been designed to minimize the amount of data that must be read from disk into primary memory to satisfy motion in any given direction. The optimal size, determined with the aid of a simulation, was 128 pixels wide by 64 lines high for a screen with standard (4:3) aspect ratio.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense Under contract MDA-903-78-C-0122, ARPA Order 3487, and by the Computer Science Research Division, Computer Corporation of America.

EXHIBIT
23

EKC 000141910

B-062

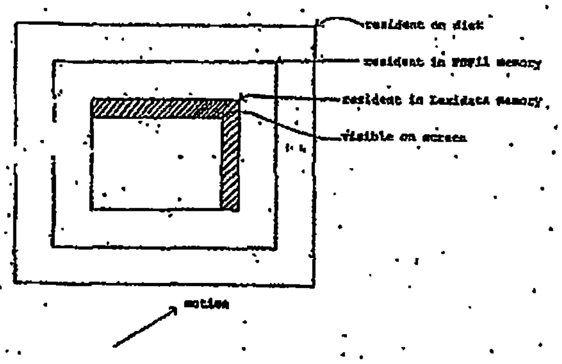
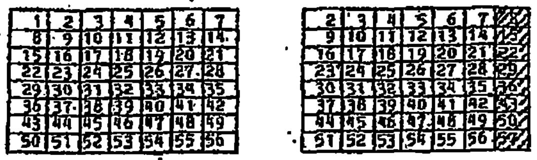


Figure 3.1 Nesting of Data

The system attempts to keep all tiles containing the information currently on the screen plus a suitable margin in core at all times. The address of these tiles are stored in an array called the tile map. This array allows any reference to a point on the displayed image to be mapped into the appropriate tile address in core. Along with the address of the tile, each element of the tile map contains the disk address where the tile is stored and a number of flag bits which indicate whether the data in the core resident tile is valid and whether it has been modified.

The tile map allows the motion control system to ensure that a suitable margin is maintained in core at all times. When scrolling has caused the displayed area to deviate from the center of the data represented by the tile map, the table is rolled — that is, the data in it is shuffled so that the data represented in the center of the tile map is once again that which is in the center of the screen. This process is illustrated for the case of motion to the right in figure 3.2. The tile addresses which have wrapped around from one margin to the other, shown as the shaded column in the figure, are flagged as being invalid. The motion control system then computes the disk addresses necessary to fill these tiles with new data corresponding to the new margin and issues the appropriate disk read requests. The disk reading process flags each tile as containing valid data when it has been read in.



1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49
50	51	52	53	54	55	56

2	3	4	5	6	7	▨ 8
9	10	11	12	13	14	▨ 15
16	17	18	19	20	21	▨ 22
23	24	25	26	27	28	▨ 29
30	31	32	33	34	35	▨ 36
37	38	39	40	41	42	▨ 43
44	45	46	47	48	49	▨ 50
51	52	53	54	55	56	▨ 57

Figure 3.2 Rolling of Title Map

The tile map is also used by a paint process and other processes which modify the image in core. After modifying the appropriate image data, such a process sets the corresponding modification bits in the tile map entry, causing the system to write the modified tiles back to the disk when they are rolled out of the map.

In order to be seen on the screen, image plane data must be resident in the frame buffer. The system endeavors to maintain a margin of data around the visible image. As the user causes the display to scroll into this margin, the system re-uses the memory left behind, writing new image data into it and thus preparing a new margin.

The re-use of the frame buffer memory requires that the display refresh logic address the memory in such a way that the memory appears to "wrap-around" from line to line and from top to bottom. The data in the display is stored as a set of contiguous scan lines. This fact, together with the limited size of the display's memory, requires that only those portions of the tiles which are visible, together with a small margin, be sent to the display from the PDP-11. These portions are composed of stripes of data which the motion software writes into the margin just ahead of the current direction of travel.

3.1 Navigator

Staging is supervised by a process known as the navigator which performs and synchronizes all the necessary activities.

The navigator is responsible for sending the display processor a command to update the point in its buffer at which it starts its display, thereby causing the image to scroll. This operation must be performed at constant intervals, approximately 30 times per second, in order for the scrolling to appear smooth and even.

When necessary, the navigator must send data to the display processor to maintain the margins of the image data which the user can scroll into. Enough new data must be sent to ensure that the display processor can always have new data to scroll on the screen.

The navigator must maintain the tile map. When necessary, it rolls the tile map, resulting in new tiles being read from the disk. This ensures that enough data in memory is available to be sent to the display processor when needed. Once again, this must be done fast enough so that there is always data in core to send to the display processor.

The navigator must also maintain status information (location, scale, etc.) for use by other processes and update the navigational aid.

A separate process is used to perform the actual disk reads and writes of the tiles. This allows the disk activity to be performed asynchronously, eliminating seek delays which would be unacceptable.

In the next section, a hardware system is described which would replace the function of the disk i/o process in the current implementation of SDMS.

4. A Database Computer for Image Management

The SDMS graphics environment is primarily comprised of images of very low complexity resulting from the automatic object description creation mechanisms in SDMS.[1] [4] However, a small amount of very high complexity is also present as the result of video input and manual "painting" in the environment. These environmental characteristics suggest the use of an image management subsystem which receives and transmits the image data in pixel map form, but which encodes the image internally for efficient storage. Naturally, the speed at which the image management system operates should be as fast as possible.

The principal modification to the SDMS architecture is shown below (figure 4.1):

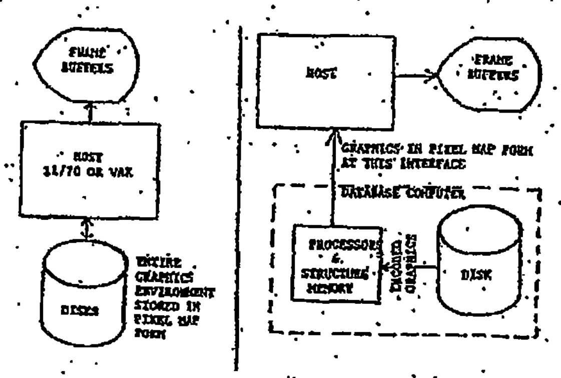

FRAME BUFFERS
HOST 11/70 OR VAX
DISKS
ENTIRE GRAPHICS ENVIRONMENT STORED IN PIXEL MAP FORM
HOST
FRAME BUFFERS
GRAPHICS IN PIXEL MAP FORM AT THIS INTERFACE
DATABASE COMPUTER
PROCESSORS & STRUCTURE MEMORY
ENCODED GRAPHICS
DISK

Figure 4.1 Database Computer Extension to SDMS

Currently, the large graphics environment is manipulated at 64 line by 128 pixel granularity. Such a subdivision is referred to as a "tile," and for the purposes of this paper, I will assume that tile size in the proposed system.

Very informal analysis shows that approximately 40 percent of all tiles in the SDMS environment are completely blank (all pixels set to the background color) and that approximately one percent of the environment is comprised of highly complex images. This analysis suggests an image storage scheme which meets the following criteria:

1. very efficient "storage" of blank tiles
2. efficient storage of low complexity images in an encoded form.
3. the potential to store extremely complex images

These criteria can be met by a database computer composed of a large disk, a moderate size medium speed random access memory (referred to as "structure memory" in the database computer literature³), and a network of dedicated purpose microprocessors.

4.1 The Disk

A large multi-platter disk, such as a ten platter 300 megabyte disk is used to store the image data. To allow parallel disk reads and writes, the disk is modified to equip each read/write head with the necessary hardware to provide concurrent operation. Image data is stored in two formats: pixel map and run length encoded.

Pixel map format is used to store highly complex images. Pixel map format is the actual storage of the numeric value of each individual pixel. This scheme allows images of unlimited complexity to be represented.

Run length encoding represents a raster segment by N color-run length pairs. This kind of encoding is far more economical than pixel map representation, but at most N-1 color changes over the raster segment can be represented.

Each 64 line by 128 pixel tile in the environment is represented on disk by one tile descriptor and zero or more pixel map line areas. A tile descriptor requires two blocks (assuming 512 byte block size) of disk storage. Each block of a tile descriptor contains 32 line descriptors in run length encoded format (figure 4.2).

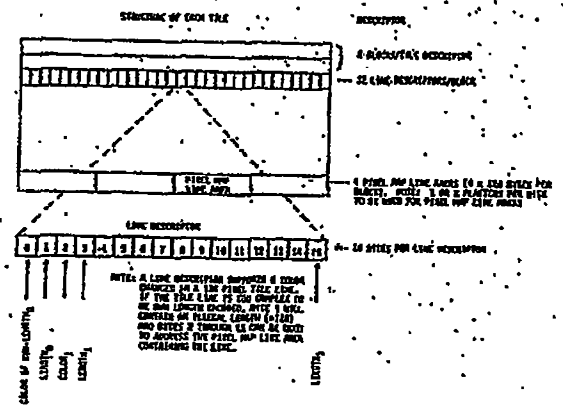


Figure 4.2

Each line descriptor is composed of eight two-byte color-run length pairs. Thus, a tile descriptor can fully represent a tile as long as each 128 pixel line incorporates fewer than eight color changes. When a line's complexity exceeds this constraint, it must be stored in pixel map form in a "pixel map line area" on the disk. When this is necessary, the line descriptor is used to contain the address of the pixel map line area containing the line.

Image data is arranged on the disk such that all tiles in a given disk cylinder and sector are vertically contiguous in the graphics environment, and all tiles in a given disk cylinder and platter are horizontally contiguous in the environment.

Each "disk cylinder high" stripe across the graphics environment is stored as a series of concentric disk cylinders uniformly distributed over the disk (figure 4.5).

One or two disk platters are reserved for pixel map line areas, stored four to a disk block (4 x 128 pixels).

Section 4.3 of this paper shows how the arrangement of image data described above helps minimize disk delay, particularly in retrieving tiles.

4.2 The structure memory

Approximately 128K of structure memory contains the following data structures:

- Blank Tile Flag array — One bit per tile indicates that the entire tile is set to the background color.

- Pixel Map Line Area Free Flag array — One bit per pixel map line area indicates that the area on disk is free.
- Tile Buffer Pool — some number of tile buffers, to allow parallel disk reads and writes, and to allow double buffered image data transfer between the host computer and disk.
- Tile Buffer Pool Directory — to maintain the status of the tile buffer pool.
- Pixel Map Line Buffer Pool — A pool of pixel map line buffers to be used in conjunction with the tile buffer pool to contain the lines of a tile which are too complex to be encoded.
- Pixel Map Buffer Pool Directory — to maintain the status of the pixel map line buffer pool.

4.3 The Dedicated Microprocessors

The major data paths connecting the dedicated processors and structure memory in the database computer is shown below (figure 4.3). A general description of the role of each dedicated processor follows:

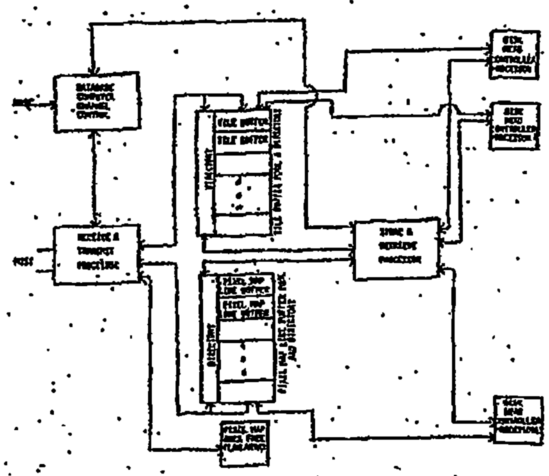


Figure 4.3

4.3.1 Receive/transmit processor. The receive/transmit processor encodes on receive and decodes on transmit, of image data.

When receiving tiles, the receive/transmit processor encodes each tile into an available tile buffer and available pixel map line buffers as necessary. If the entire tile is composed of background color, the appropriate blank tile flag is set (and the tile is effectively "stored"). Otherwise, the appropriate pixel map line area free flags are cleared, and the tile buffer pool and pixel map line buffer pool directories are updated. The update information includes that which is required by the store/retrieve processor to write the tile to disk.

When transmitting tiles back to the host, the tile buffer pool and pixel map line buffer pool directories are consulted to determine when a tile has been retrieved from disk and is ready for transmission. The transmission operation involves decoding the contents of a tile buffer and inserting the contents of pixel map line buffers where appropriate.

4.3.2 Store/retrieve processor. The store/retrieve processor schedules the 17 disk head controller processors. This processor is responsible for the management of pixel map line storage areas on the disk. The locations of tile descriptors on the disk is a fixed function of tiles' X-Y coordinates.

When storing tiles, the store/retrieve processor attempts to write tiles concurrently, if possible. That is, if two or more tiles to be written exist in the same disk cylinder and sector, they will be written in parallel by multiple disk head controller processors. The store/retrieve processor also attempts to arrange pixel map lines on the disk to minimize read times of associated tiles. This is accomplished by attempting to write pixel map lines in the same disk cylinder and sector as associated tile descriptors.

Read operations are similar to write operations subject to the overall tile retrieve strategy (see below). Read operations involve the added complexity of determining which pixel map line areas are needed for each particular tile descriptor. This information is provided by the disk head controller processors.

4.3.3 Disk head controller processors. These processors provide low-level control the logic-per-track hardware required for parallel disk reads and writes. During tile reads, they scan the contents of the tile descriptors and send the addresses of the required pixel map line areas to the store/retrieve processor.

4.3.4 Database computer channel control. The channel control is responsible for overall coordination of the database computer and for efficient operation of the tile retrieve strategy (described below).

4.4 Overall tile retrieve strategy

In general, tiles are retrieved far more frequently than they are written. This is a result of constant traversal of the graphics environment. Since speed of traversal is bound by the speed of tile retrieval, retrieval time should be carefully optimized.

Virtually all traversal-necessitated tile retrievals involve a vertical or horizontal stripe of tiles for transfer to the frame buffer display. Hence, arrangement of image data of the disk and overall retrieval strategy should be tuned to such operations. To this end, image data has been arranged such that all tiles in a given disk cylinder and sector are vertically contiguous in the graphics environment, and all tiles in a given disk cylinder and platter are horizontally contiguous in the environment. This arrangement facilitates parallel reads of vertically contiguous tiles, and minimal delay reads of horizontally contiguous tiles.

A distributed algorithm running on the channel control and store/retrieve processors attempts to perform disk reads in anticipation of the above retrieve request arrival characteristics. In particular, in response to a singular tile retrieve request, any tiles below the requested tile in the same disk cylinder and sector will be read in parallel. Then, tiles to the right of the requested tile will be read into any available tile buffers.

Tiles either below or to the right will be flushed from the buffer pools at the arrival of the next tile retrieve request; if the next requested tile is below the first, the tiles to the right will be flush, if the next request is for a tile to the right, the tiles below the first will be flushed.

Further, unless a request breaks the pattern, disk reads should anticipate tile requests in the current vertical or horizontal stripe. In particular, as tile buffer availability permits, tiles further below or to the right of the current tile should be read.

4.5 Alternative Tile Retrieval Strategy

Two further design enhancements are proposed for making the tile retrieval strategy faster and more efficient, given approximately 800k of additional structural memory to expand the tile buffer pool.

The first is to incorporate a tile map in the channel control processor. The tile map is used as an aid in managing the tile buffer pool. The tile map would have the ability to be rolled in an analogous manner to the core buffer manager in the host graphics machine. This will enable the channel control processor to anticipate requests for tiles made by the host machine by pre-fetching the tiles into the tile buffer pool. The tile map used by the channel control processor would be larger than the tile map used by the host in order to provide the margins necessary to perform "anticipatory" tile retrieval.

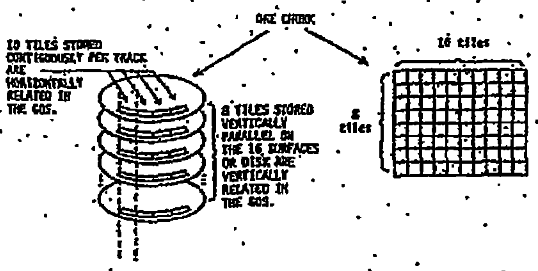
The second enhancement is intended to significantly reduce the number of disk seeks necessary to retrieve a row or column of tiles from the disk (e.g., when rolling or loading the tile map). It is conceptually possible to reduce this to one seek per roll of the tile map. Unfortunately, this optimal strategy requires large amounts of memory, making it potentially more desirable to use two or four seeks. If the seek time can be tolerated, memory requirements can be reduced substantially.

In the host graphics machine, the tile map used for staging at scales 2 and up is 9 rows by 7 columns. This requires that the smallest tile map used by the channel control processor be 11 rows by 9 columns.

If there are 16 surfaces available, each with its own read/write head, then 8 vertical tiles can be read in parallel. If the tile map used by the channel control processor is 8 rows in height, and if the tile map is in a matching alignment with the structural layout of tiles on the disk, it is possible to read all 8 tiles in parallel with only one seek.

To ensure that the tile map maintains alignment with the tile structure on disk, it is necessary to load the tile map at intervals of 8 rows of tiles. This requires a margin of 8 rows in the tile map in order to maintain the alignment. Each time the map is rolled, it is rolled by 8 rows. This reduces the number of seeks necessary for vertical motion.

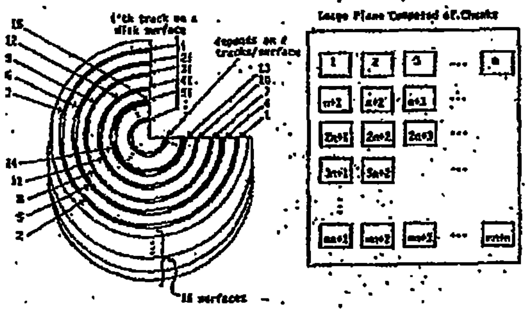
If the tiles are laid out contiguously horizontally on the same disk track, (see figure 4.4) it is possible to read a complete row of the tile map in one disk seek and one rotation time. As long as the number of tiles on a track is evenly divisible by the length of a row in the tile map, there will need to be only one disk seek per row of tiles read in for the tile map (see figure 4.5).


ONE CHUNK
10 TILES STORED CONTIGUOUSLY PER TRACK ARE HORIZONTALLY RELATED IN THE GOS.
8 TILES STORED VERTICALLY PARALLEL ON THE 16 SURFACES OF DISK ARE VERTICALLY RELATED IN THE GOS.
16 tiles
8 tiles

EACH OF THESE "STRONGLY" STRUCTURALLY RELATED (VERTICALLY AND HORIZONTALLY) BLOCKS OF TILES IS REFERRED TO AS A CHUNK. THUS A CHUNK IS 8 ROWS BY 10 COLUMNS IN SIZE AND CAN BE READ USING ONE DISK SEEK.

Figure 4.4

STRUCTURAL LAYOUT OF CHUNKS ON THE DISK SURFACE


i'th track on a disk surface
depends on # tracks/surface
Large Plane Composed of Chunks
15
12
9
6
3
14
11
8
5
2
13
10
7
4
1
16 surfaces
1
2
3
n
n+1
n+2
n+3
2n+1
2n+2
2n+3
3n+1
3n+2
mn+1
mn+2
mn+3
m+n

Depending upon the number of sectors, and the size of each sector, there may be one or more "chunks" per track. In this particular example, three chunks per track are represented. The storage of each row of chunks is uniformly distributed over the disk. This scheme of interleaving the rows of chunks minimizes the maximum seek necessary for horizontal or vertical motion.

Figure 4.5

If margins of 10 and 8 tiles are used in both X and Y respectively (10 columns by 8 rows), then whenever the tile map is rolled, 2 groups of either 8 rows or 10 columns will need to be read. Since the horizontal and vertical structural relationships will parallel that on the the disk, the two chunks of 8 rows by 10 columns can be read with only two seeks.

When a user scrolls in a diagonal direction, the tile map is always rolled in two explicit operations, first in one direction (e.g., horizontal) and then in the other (vertical). This ensures that the memory requirements to maintain the map remain constant.

4.6 Performance

Usable disk area of 250 megabytes is sufficient to store 250,000 tiles at 1K (2 x 512 blocks) per tile. Assuming the need to store one percent of the environment at pixel map detail, approximately 37,500 blocks will be required for pixel map line areas. The remaining disk capacity is about 231,250 tiles. At ten tiles per screen in the current SDMS, the capacity of the image data management system would be approximately 23,125 screens.

Reliable estimation of the speed at which the system would operate must await more careful analysis of the proposed design.

5. References

1. Herot, Christopher F. "Spatial Management of Data." ACM Transactions on Database Systems (to appear March, 1980, also available as Technical Report CCA-79-24 from the Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139).

2. Herot, Christopher F., Carling, Richard T., Friedell, Mark, Kramlich, David and Thompson, John "Spatial Data Management System: Semi-Annual Technical Report." Technical Report CCA-79-25, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

3. research and development efforts at the following sites: Case Shaded Graphics Lab., Case Institute of Technology; The Wharton School, University of Pennsylvania

4. Herot, C. F., Carling, R., Friedell, M., Kramlich, D. "A Prototype Spatial Data Management System." Proceedings of the SIGGRAPH 1980 Annual Conference.

5. Banerjee, Jayanta, Baum, Richard I. and Hsiao, David K. "Concepts and Capabilities of a Database Computer." ACM Transactions on Database Systems 10 (December 1978).

Spatial Management of Data

CHRISTOPHER F. HEROT
Computer Corporation of America

Spatial data management is a technique for organizing and retrieving information by positioning it in a graphical data space (GDS). This graphical data space is viewed through a color raster-scan display which enables users to traverse the GDS surface or zoom into the image to obtain greater detail. In contrast to conventional database management systems, in which users access data by asking questions in a formal query language, a spatial data management system (SDMS) presents the information graphically in a form that seems to encourage browsing and to require less prior knowledge of the contents and organization of the database.

This paper presents an overview of the SDMS concept and describes its implementation in a prototype system for retrieving information from both a symbolic database management system and an optical videodisk.

Key Words and Phrases: computer graphics, man-machine interaction, database query languages, graphics languages
CR Categories: 3.7, 8.2

1. INTRODUCTION

Spatial data management is motivated by the needs of a growing community of people who have to access information in a database management system (DBMS) but who are not trained in the use of such systems. The information in a spatial data management system (SDMS) is expressed in graphical form and presented in a spatial framework, so that the information is more accessible and its structure more obvious than in a conventional DBMS. In this way a user can find the information he seeks without having to specify it precisely or know exactly where in the DBMS it is stored.

A user "retrieves" data from SDMS by examining a flat surface upon which pictorial representations of the data are arranged. This approach permits many types of questions to be answered without the need for a keyboard, although a conventional symbolic query language is also provided.

The graphical data space (GDS) of SDMS is accessed through a set of three color raster-scan displays, as illustrated in Figure 1 (page 497). The leftmost of the three screens presents a "world-view" map of the entire data surface. A

The work reported herein was sponsored by the Defense Advanced Research Projects Agency under Contract MDA903-78-C-0122. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.
Author's address: Computer Corporation of America, 575 Technology Square, Cambridge, MA 02139.
© 1980 ACM 0362-5915/80/1200-0493 $00.75

magnified portion of this data surface is simultaneously displayed on the main screen in the center. The location on the data surface of this magnified portion is indicated by a highlighted rectangle on the world-view map. The user can control which portion of the data surface appears on the main display by pressing on the joystick shown in the user's left hand, in the foreground of the figure. Pressing the joystick in any given direction causes the user's magnified window to move in that direction over the data surface, and this motion is reflected in a corresponding motion of the highlighted rectangle on the world-view map.

The simultaneous presentation of a world view and a detailed view of the data surface ensures that the user is always aware of his current location in the database. It also aids in finding a region of interest in a very large database.

The data presented to the user on the main display can come from a variety of sources. The three sources described in this report are

(1) images stored as bit arrays on a digital disk,
(2) a symbolic DBMS,
(3) an optical videodisk.

Tools are provided which allow the user or the database administrator to establish connections among data from these sources.

The use of the system is best illustrated through an extended example; one is given in Section 2; Section 3 contrasts SDMS with conventional DBMSs; Section 4 compares SDMS with previous work that combines the use of computer graphics and databases; Section 5 details the use of the optical videodisk for storage and retrieval of analog video information; Section 6 describes some additional features of the system for defining the relationship between graphical and symbolic representations of data; Section 7 describes SQUEL, the query language that allows the use of typewritten queries with graphical retrieval; Section 8 gives a brief account of the progress on the implementation of the SDMS prototype; and finally, Section 9 presents some preliminary conclusions about the overall effectiveness of spatial management of information.

2. EXAMPLE

This section illustrates the use of SDMS as an interface to a symbolic DBMS.

The user in this example is examining a database of ships. This database originated as a conventional database managed by the INGRES database management system [7]. A fragment of the SHIP relation in the DBMS is shown in Figure 2.

The procedure by which the database administrator produced the graphical representation used in the example is described in Section 2.2.

2.1 Retrieval of Data

Figures 3 through 7 (pages 498–500) illustrate how SDMS is used to retrieve information from the database.

Figure 3 shows the world-view map which is presented on the leftmost of the three screens. It is an all-encompassing view of the data laid out on the graphical data surface. The map has been divided into two rows, one for each of the two countries having ships in the database. Each row is further divided into columns,

uic	nam	type	nat	ircs	beam	ready
N00001	CONSTELLATION	CV	US	NABC	130	1
N00002	KENNEDY JF	CV	US	NABD	130	1
N00003	KITTY HAWK	CV	US	NABE	130	2
N00004	AMERICA	CV	US	NABF	130	5
N00005	SARATOGA	CV	US	NABG	130	1
N00006	INDEPENDENCE	CV	US	NABH	130	1
N00007	LOS ANGELES	SSN	US	NABI	33	1
N00008	BATON ROUGE	SSN	US	NABJ	33	1
N00009	PHILADELPHIA	SSN	US	NABK	33	1
N00010	STURGEON	SSN	US	NABL	32	1
N00011	WHALE	SSN	US	NABM	32	1
N00012	TAUTOG	SSN	US	NABN	32	1
N00013	GRAYLING	SSN	US	NABO	32	1
N00014	POGY	SSN	US	NABP	32	1
N00015	ASPRO	SSN	US	NABQ	32	1
N00016	SUNFISH	SSN	US	NABR	32	1
N00017	CALIFORNIA	CGN	US	NABS	61	1
N00018	SOUTH CAROLINA	CGN	US	NABT	61	1
N00019	DANIELS J	CG	US	NABU	55	1
N00020	WAINWRIGHT	CG	US	NABV	55	1
N00021	JOUETT	CG	US	NABW	55	1
N00022	HORNE	CG	US	NABX	55	1
N00023	STERETT	CG	US	NABY	55	3
N00024	STANDLEY WH	CG	US	NABZ	55	1
N00025	FOX	CG	US	NACA	55	1
N00026	BIDDLE	CG	US	NACB	55	1
N00027	LEAHY	CG	US	NACC	55	4
N00028	YARNELL HE	CG	US	NACD	55	1
N00029	WORDEN	CG	US	NACE	55	1
N00030	DALE	CG	US	NACF	55	1
N00031	TURNER RK	CG	US	NACG	55	1
N00032	GRIDLEY	CG	US	NACH	55	1
N00033	ENGLAND	CG	US	NACI	55	1
N00034	HALSEY	CG	US	NACJ	55	1
N00035	REEVES	CG	US	NACK	55	3
N00036	ADAMS CF	DDG	US	NACL	47	3
N00037	KING J	DDG	US	NACM	47	1
N00038	LAWRENCE	DDG	US	NACN	47	1
N00039	RICKETT CV	DDG	US	NACO	47	1
N00040	BARNEY	DDG	US	NACP	47	1
N00041	WILSON HB	DDG	US	NACQ	47	1
N00042	MCCORMICK L	DDG	US	NACR	47	1
N00043	TOWERS	DDG	US	NACS	47	1
N00044	SELLERS	DDG	US	NACT	47	1
N00045	ROBISON	DDG	US	NACU	47	1

Fig. 2. Ship database.

one for each class of ship. Within each column, each ship is represented by a colored shape, referred to as an *icon*.

The color of each icon indicates the ship's readiness, one of the attributes of each ship in the symbolic database. Green indicates the highest readiness, followed by yellow, orange, and red.

The size of each icon is a function of the beam of its associated ship. This is most apparent among the Russian carriers, located in the lower left corner of the data surface.

In the section in the center of the top row, one can see a highlighted rectangle, which indicates the portion of the data surface that is shown in the magnified view presented on the main display. The position of this rectangle, and thus the portion of the data surface shown on the main display, is controlled by the joystick shown in Figure 1.

Figure 4 shows the main data display which appears on the center screen. Notice that while the position and color of the ships are the same as in the highlighted rectangle of the world-view map, the magnified view is more detailed. The icons have a more detailed shape, and beneath each shape are three text strings, which give the ship's international radio call sign, name, and commanding officer.

Given this graphical view of the symbolic database, let us see how a user would go about retrieving information on a specific ship, in this case the *Daniels*. To start, he pulls the joystick down toward himself, indicating that he wishes to move the magnified view toward the bottom of the data surface. When the *Daniels* is in the center of the screen, as in Figure 5, he releases the joystick.

Now, to get a more detailed view, he twists the joystick clockwise, instructing the system to expand the magnification of the picture shown on the screen. One instant in this zooming process is captured in Figure 6. Once the machine has magnified the image, it adds more detail, as shown in Figure 7. The outline of the ship becomes more detailed, and the space under the ship is filled with values of attributes from the symbolic database.

At this point the user has a choice of actions. He can move laterally across the data surface and examine similarly detailed views of adjacent ship icons. He can twist the joystick in the counterclockwise direction to return to the less detailed views of Figures 5 and 6. Or he can, if the database administrator has provided for it, once again twist the joystick clockwise and magnify the view in Figure 7, getting a still more detailed icon with yet more attribute values displayed.

2.2 Creation of the Graphical Data Surface

The preceding paragraphs described how one might use a graphical view of a symbolic database to retrieve some information. In order to construct such a view, the database administrator must first describe to the system how each icon should appear and then instruct the system to create a data surface of icons for selected tuples in the DBMS.

2.2.1 *Icon-Class Description.* The SDMS provides the database administrator with a tool for describing the appearance of each icon as a function of attributes of tuples in the DBMS. That tool is the icon-class description language (ICDL). It consists of a series of statements, each of which accepts some attribute value and performs some graphical operation. The ICDL that was used to generate the icons shown in the preceding example is given in Figure 8.

The *position* statement determines the placement of the icon on the data surface. In the example it maps the ship's type into x coordinates and its

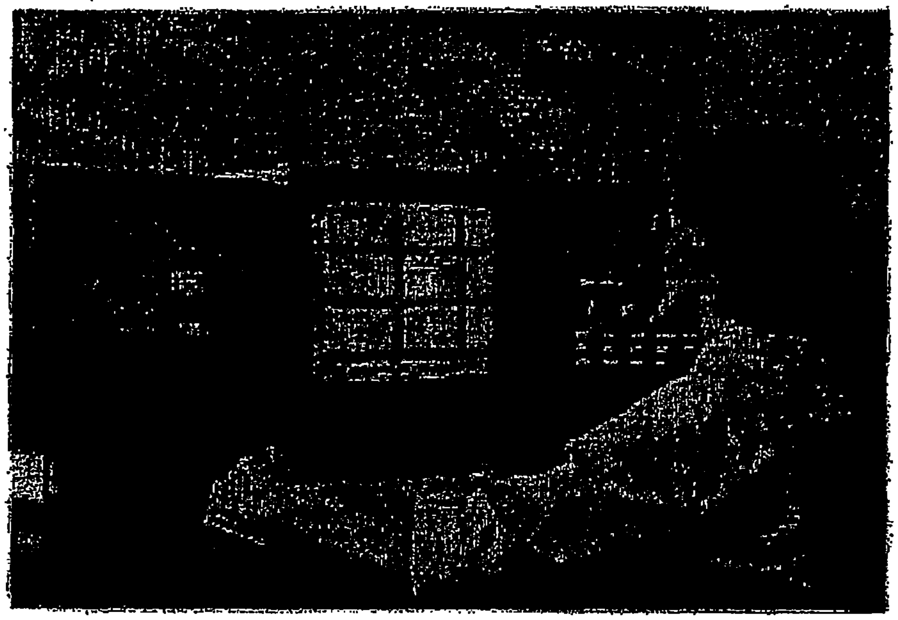


Figure 1. User Station

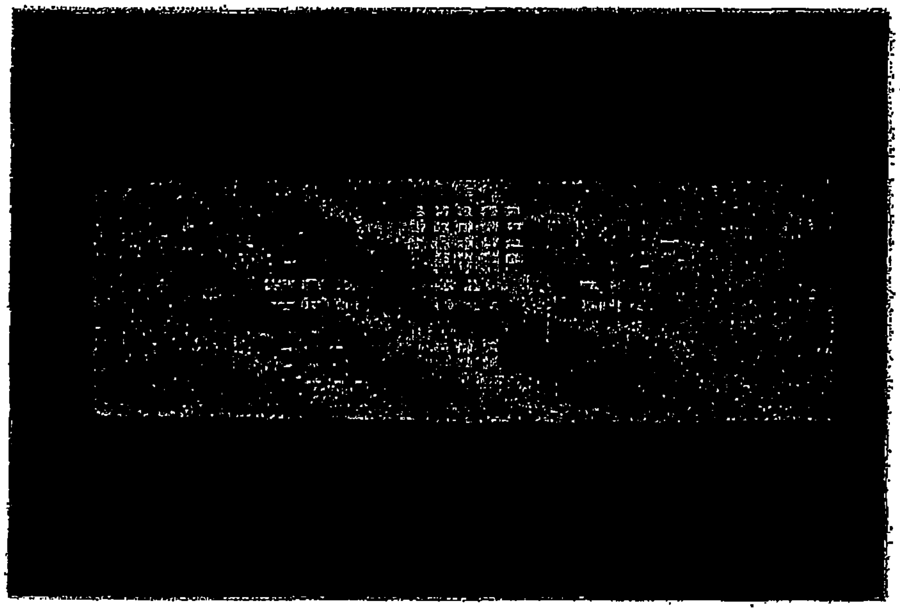


Figure 3. World-view map

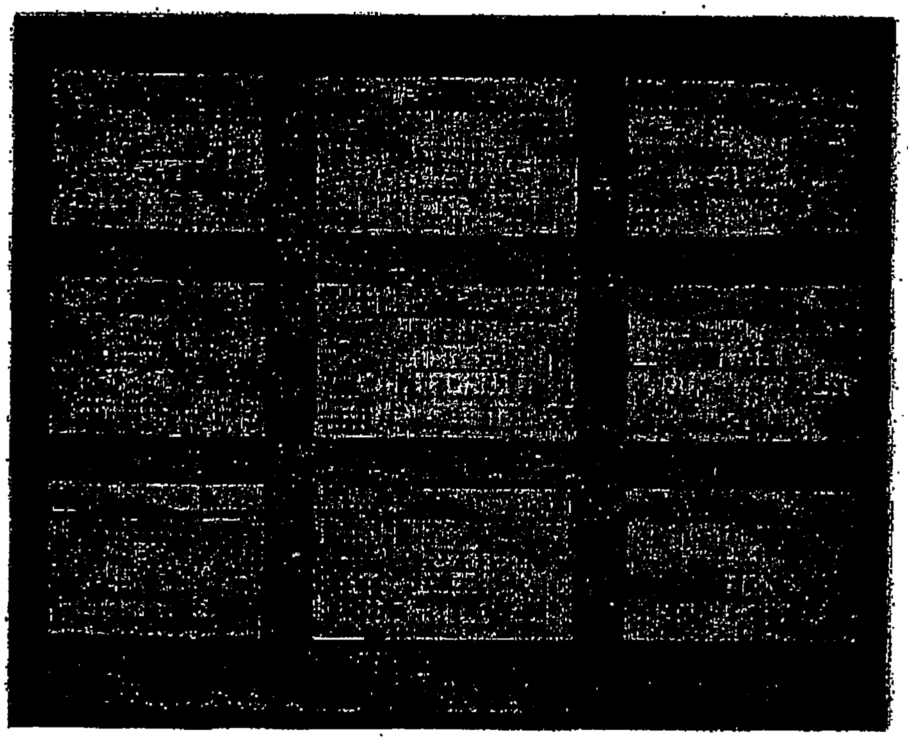


Figure 4. Magnified view of data surface

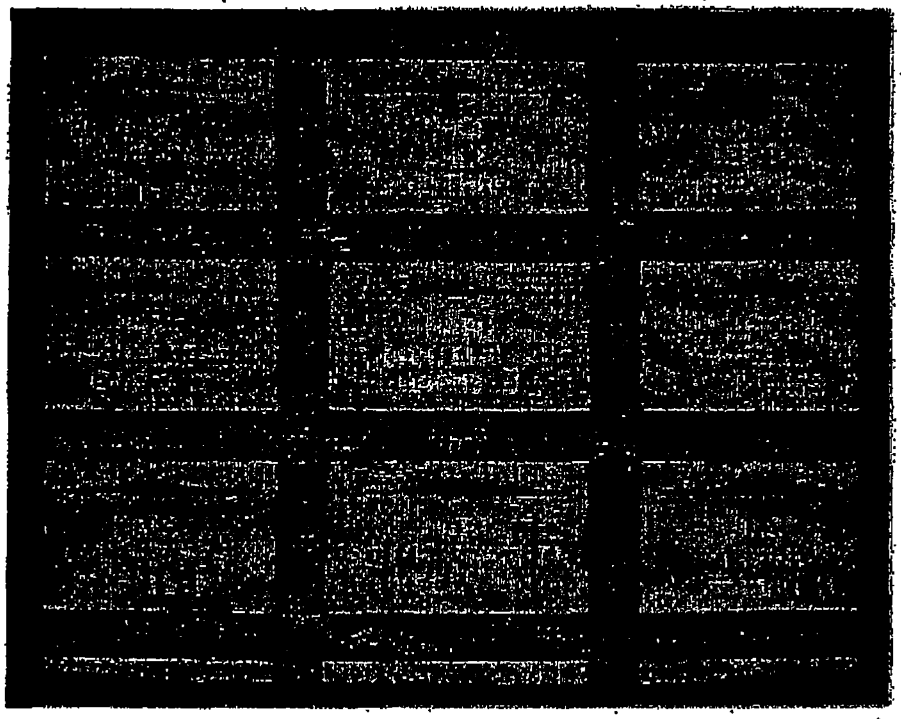


Figure 5. Magnified view with "Daniels" centered

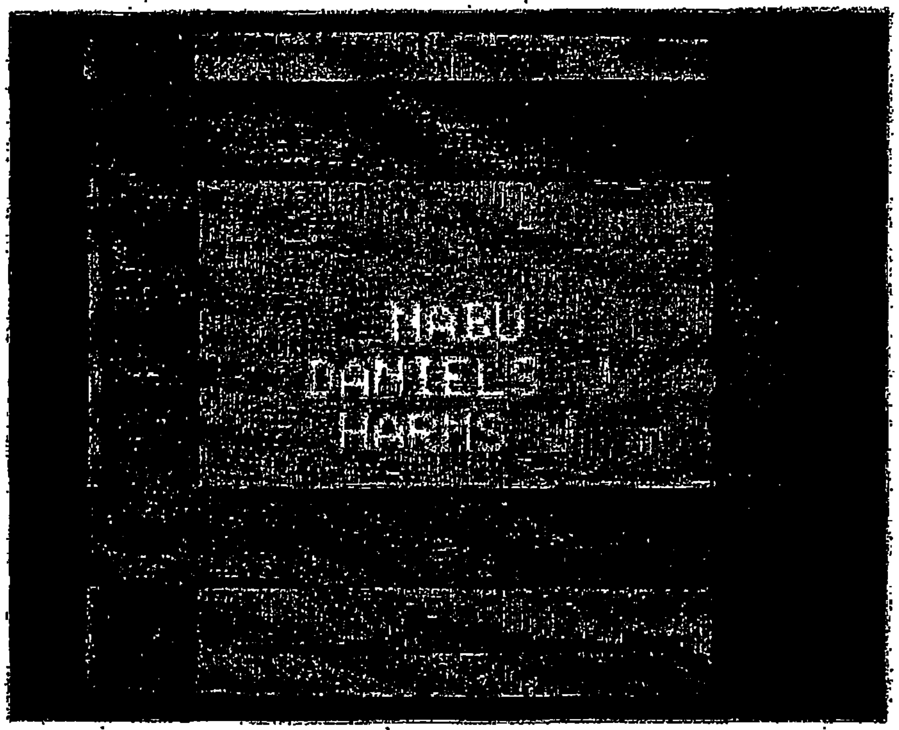


Figure 6. Zoomed in view of data surface

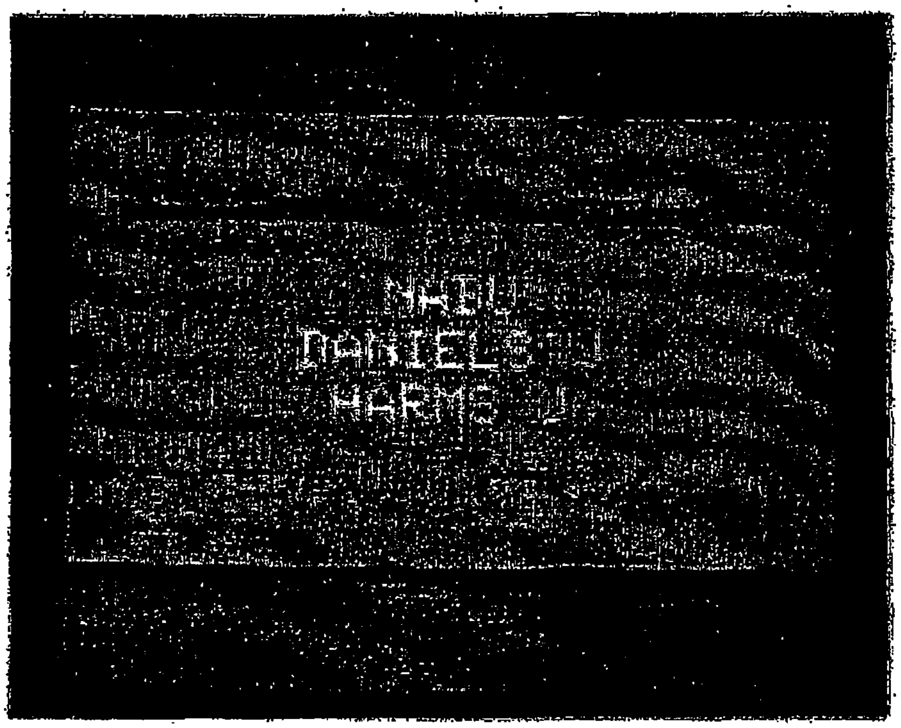


Figure 7. More detailed view of data surface

```
icon class cluster(r)
begin
maximum size is (110,60);
position is
   (case r.type   begin
                  "CV":     800
                  "SSN":    1600
                  "SSBN":   1600
                  "SSGN":   1600
                  "CGN":    2500
                  "CG":     2500
                  "CA":     2500
                  "DDG":    3200
                  "FF":     3200
                  "AGI":    3200
                  "AO":     4000
                  default:  1600
                  end,      case r.nat begin
                            "US":1200
                            "UR":2000
                            default:2500
                            end);
    template icon case r.type   begin
                                "CV":     carrier
                                "SSN":    sub
                                "SSBN":   sub
                                "SSGN":   sub
                                "CGN":    cruiser
                                "CG":     cruiser
                                "CA":     cruiser
                                "DDG":    destroyer
                                "FF":     destroyer
                                "AGI":    destroyer
                                "AO":     oiler
                                default:  cruiser
                                end;
    scale is r.beam*2 percent;
    color of region 1 is case r.ready   begin
                                        "1":green
                                        "2":yellow
                                        "3":orange
                                        "4":red
                                        default:gray
                                        end;
    attribute region r.nam from (5,16) to (70,28);
    attribute region r.ircs from (5,28) to (70,40);
    attribute region r.consm from (5,40) to (70,52);
end;
```

Fig. 8. Icon-class description.

nationality into *y* coordinates. The *template* statement specifies the shape of the icon by selecting among a set of pictures that have previously been drawn by the database administrator. The *scale* statement specifies the size of the icon as a function of the beam of the ship. The *color* statement specifies the color of each ship according to its readiness. Finally, the three *attribute region* statements place the values of the ship's name, international radio call sign, and commanding officer's name into the specified locations in the icon.

2.2.2 *The Association Statement.* Having given the rules for the appearance of each icon, the database administrator creates the graphical representation by using the *associate* statement. This causes SDMS to select particular tuples from a specified relation in the database and to create icons from them on the basis of the designated icon class description.

To generate the graphical data surface of the preceding example, the database administrator would type

associate ship using cluster

This causes SDMS to retrieve the tuples from the relation SHIP and pass them one at a time to a module which interprets the ICDL "cluster," using the values of the attributes of the tuple. For each such tuple an icon is created on the graphical data surface.

The associate statement also permits the use of a qualification to select tuples. For example, a graphical data surface containing icons for those ships having a readiness of other than 1 could be created by typing

associate ship using cluster where ship.ready != 1

This allows the combination of the capabilities of symbolic and graphic retrieval in searching for information.

3. CONTRAST WITH CONVENTIONAL DBMSs

As may be seen from the foregoing example, an SDMS offers several advantages over conventional keyboard-oriented DBMSs, including those offering natural language or English-like user interfaces. In this section six such advantages are discussed:

(1) Motion through the database is simple and natural.
(2) The database is its own data dictionary.
(3) The presentation of the data encourages browsing.
(4) The placement of the data can convey information.
(5) Graphics can be used to convey information.
(6) The system can accommodate many unique data types, such as photographs.

3.1 Motion Controls

The joystick of SDMS provides a simple and natural means of moving through the database. By using one control, the joystick, the user can explore the entire database. On the other hand, symbolic query languages, such as QUEL [4], require the use of a special syntax and semantics. Even in natural-language systems, where the syntax is widely known and the semantics relatively intuitive, the user must learn the structure and contents of the database before he can find anything. By contrast, the data in an SDMS can be displayed in a manner that makes the contents and structure readily apparent and does not require any prior knowledge of the structure of the database in order to retrieve information from it.

The level of abstraction at which the data are examined is similarly easy to control. Twisting a knob zooms the picture and thus specifies the level of detail which is displayed, allowing the user to see global attributes, such as the number of objects in a set, without the burden of utilizing separate functions for aggregation. In addition, if the database administrator has set up more elaborate abstractions, these can be invoked by the user by the same simple mechanism, requiring no additional knowledge on his part.

When dealing with very large databases, this control of detail makes it possible for the user to see the entire database on the world-view display, even if there are more elements in the database than can occupy discrete positions on the television screen. The database administrator need merely define an abstraction which groups the data elements together in some suitable manner. For example, a database of ships might be grouped according to hull type. The more abstract views of the GDS would then display the groups instead of the individual elements, allowing the user to see the entire database and move quickly to the location of some particular area of interest. Once he had centered that area on the screen, twisting the knob would cause the individual elements to appear.

3.2 The Graphical Data Space As Its Own Data Dictionary

A conventional DBMS requires the use of a data dictionary to inform the user of the structure of the database. Even natural-language user interfaces suffer from the problem of educating the user as to what queries may be answered from the information contained in the database. In contrast, the graphical data space of SDMS is its own data description. Rather than specify a relation and attribute name, the user merely traverses the data surface until he reaches the desired information, at which point the data are laid out in front of him.

Data types are shown by example. As there will be many such examples displayed on the data surface, they serve not only to display the data type that is permitted, but also to reveal the values of the data that are typically used. This property informs the user of the ranges of the data or the shades of meaning often obscured by attribute names such as "remarks" or "comments." Since the values of such attributes are displayed on the screen, the uses made of such fields may be readily ascertained.

3.3 Browsing

The user of an SDMS is almost always presented with a display that gives him more information than he immediately needs. Within this presentation, finding the required information is facilitated by the distinctive visual qualities that can be imparted to the data. At the same time, the unsolicited surrounding data make it possible for the user to browse through the database. This is a difficult activity in a conventional database system, where every piece of data must be requested explicitly. While a small database may be printed out and examined, the lack of any mechanism for placing related data together would make such a technique impractical for very large databases. Likewise, it would be very tedious to submit repeated queries, and such a technique would be ineffective if the user were not already familiar with the contents and structure of the database.

In contrast, SDMS allows the database administrator to arrange the information in the database according to any chosen attributes. Once the user has positioned his window in the vicinity of the data being sought, he can browse through the surrounding area, letting the appearance of the icons determine where he focuses his attention.

3.4 Using Icon Position to Convey Information

The placement of a particular icon can be used to aid in recalling a particular datum, in much the same way that a person finds some needed information in his office by recalling where he put the piece of paper on which it was written.

The placement of an icon may also convey information directly. In the example of the preceding section the location of each ship indicated its nationality and type. A personnel database could be arranged according to seniority or salary. Each organization could be displayed in a separate part of the GDS, allowing the user to select whatever arrangement suited his specific query. He could then observe the world-view map to get an overall picture, such as the number of items in each category, and he could move his magnified window over some particular area to look at exceptional, extreme, or average values.

3.5 Graphic Representations

Graphic representations are often the most vivid means of conveying information, especially for aiding in the perception of trends in large aggregations of data. The most familiar forms of this technique are the histogram and graph, where numerical data are displayed in graphical form. The graphical output facilities of SDMS make such display of numerical data possible as a natural extension of the system.

Graphical representations may also be used to advantage in displaying trends in nonnumerical data. For example, a database of ships could be displayed against the background of a map, with the wake behind each ship indicating its speed and direction. With such a display it is easy to spot trends that otherwise would be hard to formulate into symbolic queries, such as a trend in a large number of ships heading for the Middle East.

3.6 New Data Types

An SDMS is not restricted to data that originate as numbers and character strings. The raster-scan output devices and digital storage of graphical data provide a natural means of storing images such as photographs. The prototype SDMS provides an interactive graphical editor, which allows a user to annotate a data surface or, without requiring any special artistic talent, to create diagrams and illustrations.

The prototype implementation also includes an optical videodisk, which can be controlled through SDMS to provide storage for a very large number of video images.

4. PREVIOUS WORK

The use of graphics to manipulate a database dates back to the earliest days of computer graphics. In 1963 Ivan Sutherland's SKETCHPAD [6], one of the first computer graphics programs ever written, illustrated how interactive graphics could be a natural mechanism for creating and manipulating geometric descriptions. Spatial data management makes the mechanism of graphical manipulation and display available to the larger application of dealing with nongeometric data. The icon-class description mechanism of SDMS allows data to be displayed spatially even when those data contain no explicit spatial information. Thus SDMS is not limited to geographic or geometric data but can spatially display any data for which the database administrator can define a suitable transformation. This transformation can be accomplished within the icon-class description language without resorting to an ad hoc "escape" to a separately compiled programming language, which is often required in other systems.

Since SKETCHPAD there have been many computer systems that have exploited the interaction of graphics and databases, but unlike SDMS or SKETCHPAD, which use graphics as a means of interacting with a database, these systems typically provide a DBMS as the front end and produce graphics as the result.

One well-known system in a research environment [9] provides the user with a relational database in which he can store graphical descriptions. These descriptions may contain mixtures of graphical primitives and references to previously defined objects, giving the user a powerful picture-building facility. By using an interactive program that edits the database, the user can easily alter his picture. The authors show how this technique could be used to combine graphical and symbolic data in service of such applications as an illustrated catalog of machine parts.

While the SDMS prototype provides similar facilities, extending them to include hand-drawn raster data, the emphasis of the system is not on making pictures but on using those pictures to access a database. Although the current implementation of the system does use a DBMS in generating its graphics, the user is encouraged to employ this mechanism as a means of accessing his own symbolic data. The graphics mechanism is superimposed on the DBMS and therefore requires no modification of the user's original symbolic data, allowing SDMS to be used for accessing an existing (possibly shared) database. Since the display mechanism is external to the database, it is possible to define multiple graphical views of the same data.

Another database system that employs graphics to retrieve information is CUPID [5], which allows the user to formulate queries by manipulating graphical symbols. Unlike SDMS, where the graphical symbols represent the complete contents of the database, the symbols in CUPID represent the various primitive operations that can be performed on the data. CUPID is therefore a graphical query language rather than a graphical view of the data and cannot be directly compared to SDMS.

A database retrieval mechanism to which SDMS does invite comparison is Query by Example (QBE) [10]. A user of QBE enters his query by showing the form of the intended result. The assumption of this approach is that such a form is easier for the novice user since it does not require him to make the mental transformation between the different forms of a query language and the result of the query. SDMS shares that assumption and carries it a step further by eliminating the query step altogether and allowing the user to see the actual data.

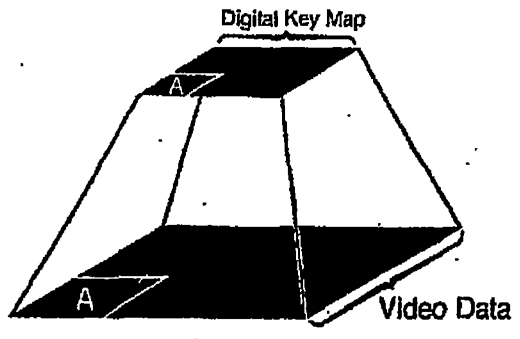


Fig. 9. Spatial data.

Furthermore, the data in SDMS are represented spatially and graphically, making trends in the data and relationships among the individual elements more visible.

5. OPTICAL VIDEODISK

The Computer Corporation of America's SDMS prototype incorporates an optical analog videodisk player. Optical videodisks have a capacity of 54,000 images, which can be viewed either as discrete photographs or in sequence as a motion picture. The same number of images stored on a conventional magnetic digital disk would require 10^{11} bits, or 100 3330-type disks.

The spatial, graphical access to information of SDMS provides a natural mechanism for retrieving spatial or graphical information stored on the optical videodisk. Two approaches are employed for accessing videodisk images.

Inherently spatial data, such as a map, are divided into sections small enough to be stored as individual video frames. The user selects a particular frame through the use of a digitally stored map, as shown in Figure 9. He can traverse this data surface as he would any other graphical data surface. When he zooms in to get a closer look, the system displays the videodisk image from the corresponding position.

A second method of accessing videodisk data is used for accessing photographs that are not spatially related, as in the case of photographs of employees in a personnel database. In such a case, illustrated in Figure 10, the user once again begins with a digitally stored graphical data surface. This surface is populated with discrete icons, one for each frame to be accessed. By zooming in on a particular icon, the user can see the associated frame. A variation on this scheme allows one icon to be associated with a sequence of videodisk images. When the icon is selected, that sequence may be examined as a motion picture or as individual photographs, with the user specifying the location in the sequence where he wishes to begin.

The mixture of analog and digital video in SDMS also makes possible the editing of data stored on the read-only videodisk. As the output of the videodisk can be directed to the center display screen at the same time that it is displaying digitally stored information, the two can be superimposed, providing a means of overlaying analog video with digital annotations. Alternatively, the analog video may be digitized and operated upon as bit-array data.

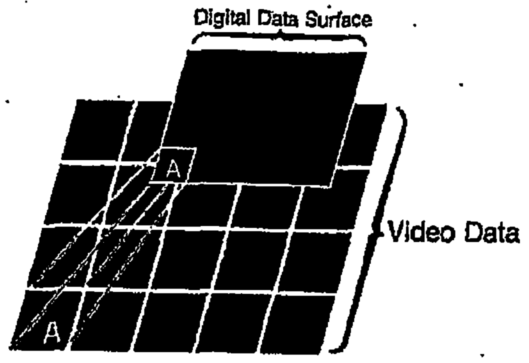


Fig. 10. Photographic data.

6. DEFINING GRAPHICAL VIEWS

This section provides additional details on the relationship between symbolic and graphical representations of data in SDMS. It introduces several features not mentioned in the example of Section 2.

6.1 Dual Representation of Data

6.1.1 *Entities and Icons.* The SDMS allows the tuples of a relation in a symbolic database management system (DBMS) to be represented in a graphical data space (GDS).[1] The graphic representation of a tuple is an *icon*. Those tuples having a graphic representation are distinguished from other tuples and are called *entities*. Each entity may be represented by more than one icon. Conversely, each icon may be a representation of more than one entity.

A goal of SDMS is to allow a user to refer to data via graphical methods, such as spatial search, or via symbolic methods, such as using a QUEL retrieval statement. To achieve this, SDMS provides a mapping between each entity and its icon. This mapping is provided via a *link* which logically connects the two. When an entity and icon are linked, a selection of one implies a selection of the other. In the query language of SDMS, the two selection methods may be combined through the use of links.

Links are created by *associating* a tuple with an icon, which makes the tuple an entity. There are two types of associations in SDMS. The first type, the *specific association*, links a specific tuple to an existing icon. It is most useful for recording symbolic information about visual data, such as entering a description of the subject matter of a photograph.

The second type, the *class association*, generates new icons for one or more tuples in a relation. The class association was used to create the graphical view of the ship's database, described in Section 2.

Specific associations are most useful for entering symbolic data for locating information that is inherently graphical, while class associations are most useful for creating graphical data for locating information that is inherently symbolic.

[1] In the Computer Corporation of America implementation of SDMS, the DBMS used is INGRES, a relational database system [4]. The query language of INGRES is called QUEL.

The class association eliminates the need for explicitly creating and linking an icon to each tuple by providing an automated mechanism for doing so.

6.1.2 *Specific Associations.* A specific association between a tuple and an icon creates a link between the two. The tuple must already exist in some relation at the time of the association. The icon must also exist in the GDS at the time of the association. Such icons will often be photographs or manually constructed diagrams. By creating tuples and linking them to such an icon, the user can then locate specific icons by the use of symbolic queries. For example, a database of paintings could be accessed by means of the artist, year, nationality, genre, etc., of a particular work.

Specific associations may also be used in conjunction with subicons (described in Section 6.4) to allow a class association to specify individual photographs. For example, the icons for the employees in a personnel database could each include the employee's photograph.

6.1.3 *Class Associations.* The class association is the principal tool for connecting the GDS to the DBMS. It is intended primarily as a tool for the database administrator (DBA).

A class association between a relation and the GDS causes the creation of icons that graphically represent the tuples of the relation. The icons are placed in the GDS. It is possible to associate only a subset of the relation by supplying a qualification that determines which tuples are to be represented. It is also possible to have the placement of the new icons restricted to a particular rectangular region of the GDS. The effects of such an association are as follows.

(1) For each tuple in the relation, an icon is created and inserted into the GDS. If a qualification was supplied, icons are created for only those tuples which pass the qualification.
(2) Each of these tuples and their corresponding icons are linked. The system will maintain the correspondence between the two as the database is updated.

When a class association is made, an icon-class description that tells exactly how to draw each icon may be specified. The icon class is essentially a picture in which certain parameters may vary each time the picture is drawn. These parameters include size, shape, color, position, and text strings. The values of these parameters may depend on the data in the entity being represented. Icon classes are discussed in Section 6.2.

When a class association is made, an explicit data dependence is established between each entity and its corresponding icon. This is manifested at first when the system draws these icons such that they reflect the data in the entity. Second, the data dependence serves to ensure that the icons that are created owing to a class association will *always* reflect the tuples they represent, until the association is explicitly broken. Hence, if an entity is updated, its corresponding icon is updated. If an entity is deleted, its icon is erased. If new tuples are added to the relation, they become entities if they pass the qualification, and new icons are created to represent them. The result is that the GDS always contains a representation for the given relation. The GDS serves as an alternate way to view the relation.

6.2 Icon Classes

An icon class is used in conjunction with a class association. While specific and class associations are relatively simple and may be performed by ordinary SDMS users, the icon-class description language (ICDL) is more complex and is meant to be utilized primarily by the database administrator.

An icon-class description consists of a series of statements that specify the appearance of the icon. Each statement performs some graphical operation, such as selecting a template picture, coloring some region, or inserting some text. The values of arguments to an ICDL statement may be taken from attributes of tuples retrieved from INGRES.

An icon constructed from an icon class may have its appearance defined at several levels of detail. This allows the zooming effect, as illustrated in Section 2, by which a user sees a more detailed version of an icon by magnifying it. The bit-array description for a single level of detail is referred to as an *image plane*. The pictures for each image plane originate as *templates*, which are simply pictures drawn by the database administrator on a special image plane reserved for that purpose. For example, the templates for ships might be drawn as follows: At the topmost (least detail) image plane the ship appears as a small rectangle. The second image plane has a rough silhouette with the ship's name and radio call sign beneath it. The third (most detailed) image plane shows some of the ship's superstructure, and the ship's name, call sign, beam, draft, speed, etc., appear beneath the picture. With such a description, a user who zooms in on such an icon will get more detail as he gets closer.

Continuing with this example, we may want different drawings for different types of ships. An icon class may include a rule for selecting among several templates.

The parameters of an icon that may be controlled by an icon-class description are

(1) *Maximum size*: The maximum size of each icon may be specified to prevent any one of them from becoming too large. Since relative size is a picture parameter, it is a good protection.

(2) *Choice of picture*: There may be several different pictures that serve as templates, depending on the data in the tuple. In the example in Section 2.1, carriers had a different picture from destroyers. If more than one picture is given, a rule for deciding which picture to use must be provided.

(3) *Target position*: The target position specifies the exact location for an icon within the GDS. SDMS does not let icons overlap, so if an icon can fit at the target position, it will be placed there. Otherwise, a location close to the target position is used.

(4) *Color*: Any region of the template may be colored. In the example, the readiness determined the color of each ship.

(5) *Size*: The template given for the description of an icon class may be scaled. In the example the size was a function of the beam of each ship. The default size is the size of the original template, up to the maximum size for the icon class.

(6) *Orientation*: The picture may be oriented by some arbitrary angle. The default is the orientation of the original template.